Apple Assembly Line

$1.80

Volume 4 -- Issue 11 August, 1984

In This Issue...

New Cross Assemblers Available

We have recently added three new cross assemblers to the S-C Macro family.

```
Intel Z-8...................$32.50
General Instruments 1650....$50.00
General Instruments 1670....$50.00
```

Unlike previous cross assemblers, which were based on Version 1.0 of the S-C Macro Assembler, these are based on Version 1.1. This means 80-column support for the Videx, STB-80, and Apple //e-//c 80-column, as well as standard 40-column. It also adds certain directives and fixes some problems which were in version 1.0.

We have also been hard at work generating Version 2.0 of the S-C Macro Assembler. It will be ready soon, complete with a brand new manual. It will support all the new opcodes and address modes of the 65C02, 65802, and 65816 processors. Owners of older versions of the S-C Assemblers will be able to upgrade for a very reasonable fee.

18-Digit Arithmetic, Part 4................Bob Sander-Cederlof

This month we will look at two output conversion routines. The first one always prints in exponential form, while the second one allows setting a field width and number of fractional digits. The routines are written so that the output string may either be printed or fed to an Applesoft string variable.

Let's assume that the value to be printed has already been loaded into the DP18 accumulator, DAC. Lines 1230-1270 describe DAC as a 12-byte variable. The exponent is in the first byte, DAC.EXPONENT. It has a value from $00 to $7F:

```
$00 means the whole number is zero
$01 means the power of ten exponent is -63
$3F means 10^-1
$40 means 10^0
$41 means 10^1
$7F means 10^63
```

The 18 digits of the number, plus two extension digits, are in the next ten bytes in decimal format (each digit takes four bits). The extension is zeroed when you load a fresh value into DAC, but after some computations it holds two more digits to guard against roundoff and truncation problems.

The sign of the number is stored in DAC.SIGN: if the value in DAC.SIGN is from $00 to $7F, the number is positive; if from $80 to $FF, the number is negative.

If you have been following the DP18 series from the beginning, and typing in all the code (or getting it from the Quarterly Disks), then you will realize that to integrate each installment takes some work. In order to print the sections separately, and have them separately readable, I must repeat some variable declarations. The listing this month refers to two previously printed subroutines, DADD and MOVE.YA.ARG. These are simply equated to $FFFF in lines 1030 and 1040, so that the code will assemble. If you really want it to work, you have to remove those two lines and include the code for the subroutines. The fact that three installments have already been printed also somewhat restricts me, because even if I see possible improvements I must be careful not to contradict the code you already have.

Quick Standard Format Conversion

The subroutine QUICK.FOUT which begins on line 1600 converts the contents of DAC to a string in FOUT.BUF in the format

```
sd.ddddddddddddddddEsxx
```

The first s is the sign, which is included only if negative. The d's are a series of up to 18 significant digits (trailing zeroes will not be included). The letter E is always included, to signify the power-of-ten exponent field. The letter s after the E is the sign of the exponent: it is always included, and

```
S-C Macro Assembler Version 1.0.....................................$80
S-C Macro Assembler Version 1.1..................................$92.50
Version 1.1 Update...............................................$12.50
Source Code for Version 1.1 (on two disk sides)....................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code)....................$20
S-C Cross Reference Utility (with complete source code)..............$50
DISASM Dis-Assembler (RAK-Ware).......................................$30
Source Code for DISASM..................................additional $30

S-C Word Processor (with complete source code).......................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk..............................$15

(All source code is formatted for S-C Macro Assembler Version 1.1.  Other
assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks...........................................each $15
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983    QD#14: Jan-Mar 1984    QD#15: Apr-Jun 1984

AWIIe Toolkit (Don Lancaster, Synergetics)...........................$39
Quick-Trace (Anthro-Digital)...........................(reg. $50)  $45
Visible Computer: 6502 (Software Masters)..............(reg. $50)  $45
ES-CAPE:  Extended S-C Applesoft Program Editor......................$60
Amper-Magic (Anthro-Digital)...........................(reg. $75)  $65
Amper-Magic Volume 2 (Anthro-Digital)..................(reg. $35)  $30
"Bag of Tricks", Worth & Lechner, with diskette.........($39.95)  $36
Aztec C Compiler System (Manx Software).................(reg. $199) $180

Blank Diskettes (Verbatim)............2.50 each, or package of 20 for $45
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each...................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
                                                        or $25 per 100
   These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers...........................  6 cents each
ZIF Game Socket Extender (Ohm Electronics) ..........................$20
QuikLoader EPROM System (SCRG)...............................($179) $170

Books, Books, Books........................compare our discount prices!
    "Apple ][ Circuit Description", Gayler..................($22.95)  $21
    "Understanding the Apple II", Sather....................($22.95)  $21
    "Enhancing Your Apple II, vol. 1", Lancaster............($15.95)  $15
                 Second edition, with //e information.
    "Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95)  $20
    "Incredible Secret Money Machine", Lancaster.............($7.95)   $7
    "Beneath Apple DOS", Worth & Lechner....................($19.95)  $18
    "What's Where in the Apple", Second Edition.............($19.95)  $19
    "6502 Assembly Language Programming", Leventhal.........($18.95)  $18
    "6502 Subroutines", Leventhal...........................($18.95)  $18
    "Real Time Programming -- Neglected Topics", Foster......($9.95)   $9
    "Microcomputer Graphics", Myers.........................($12.95)  $12
    "Microcomputer Design & Troubleshooting", Zumchak.......($17.95)  $17

We have small quantities of other great books, call for titles & prices.
Add $1.50 per book for US shipping.  Foreign orders add postage needed.

         *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
         ***                 (214) 324-2050                   ***
         *** We accept Master Card, VISA and American Express ***
```

will be either + or -. The xx is a two-digit exponent, and both digits will always be included. The decimal point will be included only if there are non-zero digits after it. If the number is exactly zero, the string in FOUT.BUF will be simply "0". Here are some more examples:

value	string
1000	"1E+03"
.001	"1E-03"
-262564478.5	"-2.625644785E+08"

Two processes are involved in converting from DAC to FOUT.BUF. One is the analysis of the DAC contents; the other is the process of storing sequential characters into FOUT.BUF. The latter process is handled in most cases by the subroutine at lines 3720-3820. Entry at STORE.CHAR stores the contents of the A-register in the next position in FOUT.BUF, and increments the position pointer (INDEX). Entry at STORE.DIGIT first converts the value in the A-register to an ASCII digit by setting the high nybble to "3". (The digits 0-9 are $30-$39 in ASCII.)

QUICK.FOUT begins by setting INDEX, the FOUT.BUF position pointer, to 0. At lines 1630-1700 the special case of the value in DAC being exactly zero is tested and handled. If the value in DAC is zero, then DAC.EXPONENT will be zero. (This is a convention throughout DP18, to simplify making values of zero and testing for them.) If the value is zero, ASCII zero is stored in FOUT.BUF, followed by a terminating $00 byte.

If the value is not zero, the next job is to check the sign of the value. Lines 1710-1740 insert a minus sign in FOUT.BUF if the value is negative.

Lines 1760-1910 pull out the 18 digits of the mantissa from DAC.HI through DAC.HI+8. The extension digits are ignored. The code here looks an awfully lot like a routine to convert from hex to ASCII, ignoring the possible hex digits A-F. That is because the digits are four bits each, and ARE like hex digits. Lines 1830-1860 insert the decimal point after the first digit.

Lines 1930-2020 look at the formatted number in FOUT.BUF and trim off the trailing zeroes. If all digits after the decimal point are zero, the decimal point is trimmed off too. If you would rather that QUICK.FOUT always printed exactly 18 digits, trailing zeroes and all, you could cut out these lines.

Lines 2040-2290 format the exponent field. First the letter E is installed in FOUT.BUF. Then lines 2060-2120 install the exponent sign. There is a little adjustment here due to the fact that the value in DAC is in the form ".DDDD" times a power of ten, and we are converting to "D.DDD" times a power. That means the exponent in DAC.EXPONENT is one larger than we will print. The DEY at line 2080 adjusts for this offset.

Lines 2130-2180 get the absolute value of the exponent by removing the $40 bias and taking the 2's complement if the

result is negative. Lines 2190-2290 convert the binary value of the exponent to two decimal digits, and insert them into FOUT.BUF. Lines 2300-2310 terminate the FOUT.BUF string with $00.

Once the value has been converted to a string in FOUT.BUF, we can either print it or put it into an Applesoft string variable. The subroutine QUICK.PRINT which begins at line 1370 calls QUICK.FOUT and then prints the characters from FOUT.BUF.

Fancier Formatted Conversion

The second conversion routine, which begins at line 2350, allows you to specify the number of digits to display after the decimal point, and the number of characters in the output field. The value will be formatted into the field against the right end, with leading blanks as necessary to fill the field. The value will be rounded to the number of digits that will be converted. If you are familiar with the FORTRAN language, you will recognize this as the "Fw.d" format. W is the width of the field, and D is the number of fractional digits. Here are some examples:

```
     W  D   value   string
     12 5   1234.56 "  1234.56000"
     12 1   1234.56 "      1234.6"
```

As before, the output string will be stored in FOUT.BUF in ASCII code, terminated by a $00 byte. If the value will not fit into the W.D field you specify, the entire field will be filled with "*" characters.

As listed here, I have set FOUT.BUF as a 41-byte variable. This means the maximum W is 40, leaving room for the terminating $00 byte. If you want longer conversions, simply change line 1060.

FOUT expects the W and D parameters to be in the A- and Y-registers, respectively. Lines 2380-2460 check W and D for legality. If W is larger than FOUT.BUF.SIZE-1, then it is set to that value. We don't want to store converted characters beyond the end of FOUT.BUF! Then if D is larger than W-1, it is pruned back.

Lines 2480-2540 initialize various variables used during the following conversion. Once again, INDEX will point to the position in FOUT.BUF. I could probably have economized some in the use of variables by re-using the same variables for different purposes, but I wanted to keep them separate to make it easier to code and debug.

Line 2560 calls ROUND.DAC.D to round the value in DAC to D digits after the decimal point. This boils down to adding .5 times 10 to the D power to the value in DAC. ROUND.DAC.D, at lines 3860-4000, does just that. First the rounding number is built in ARG, then DADD adds ARG to FAC.

Lines 2570-2610 store a minus sign into SIGN.CHAR if the value in DAC is negative. SIGN.CHAR was initialized to $00 above. If the sign is negative, line 2590 will increment SIGN.SIZE. SIGN.SIZE will either be 0 or 1, and will be used later in determining how many leading blanks are needed. We cannot store the sign character into FOUT.BUF until the leading blanks have been stored.

Lines 2630 to 2710 compute how many digits will be printed before the decimal point (NO.LEADING.DIGITS), and how many zeroes before the first significant digit after the decimal point (NO.LEADING.ZEROES). If the power-of-ten exponent was negative, there will be no leading digits and some leading zeroes; if positive, there will be some leading digits and no leading zeroes. For example,

```
     .2345E-5     .000002345     5 leading zeroes
     .2345E+3     234.5          3 leading digits
```

What if the exponent is more than 18? This would mean more digits might be extracted from DAC than exist, so lines 2730-2790 limit NO.LEADING.DIGITS to 18. NO.INTEGRAL.ZEROES takes up the slack, to print any necessary zeroes between the last significant digit before the decimal point, and the decimal point. For example, if W=25 and D=2, and the value is .1234E+20, we will get NO.LEADING.DIGITS=18 and NO.INTEGRAL.ZEROES=2:

```
        "  12340000000000000000.00"
```

Lines 2810-2870 now calculate the total number of non-blank characters which will be required: one for sign if the sign is negative, all the leading digits and integral zeroes before the decimal point, one for the decimal point itself, and D fractional digits. (Just now I noticed that I could have saved two bytes and two cycles by changing line 2810 from CLC to SEC, and eliminating the ADC #1 at line 2860.)

Lines 2890-2920 compute how many significant digits of fraction will be needed. You specified D digits of fraction, but only DD of them will come from the value in DAC. This will be less than D if there are any leading zeroes.

Lines 2940-2970 check whether the converted number can fit in a W-wide field. If not, Lines 3370-3400 fill the field with stars and exit.

Lines 2980-3030 compute how many leading blanks will be needed to right justify the number in the W-field. There is some hopscotch here because we are going to put "0." in front of numbers that have no integral digits.

At long last, we are ready to begin string characters in FOUT.BUF. Lines 3050-3070 store the leading blanks. A subroutine STORE.N.CHARS does the dirty work. STORE.N.CHARS (lines 3670-3710) expects the character to be stored in the A-register, and the count in the Y-register. It also expects that the Z-status is set according to the count in Y. Thus, if

the count is zero, the subroutines returns immediately without storing any characters.

STORE.N.DIGITS, at lines 3440-3620, is quite similar to STORE.N.CHARS. Once again, the count must be in the Y-register and the Z-status should reflect the value in Y. Digits are picked out of the value in DAC using an index DIGIT.PICKER, and stored into FOUT.BUF using STORE.DIGIT.

Lines 3090-3110 store the sign if it is negative. Lines 3120-3210 print whatever digits are needed before the decimal point. This will include leading digits (if any) and integral zeroes (if any), or simply one zero (if neither of the other).

Lines 3230-3320 store the fractional part. This includes the decimal point, leading fractional zeroes (if any), and fractional digits (if any).

Finally, lines 3340-3350 store a terminating $00 at the end of the string in FOUT.BUF.

A subroutine called FORMAT.PRINT at line 1450 calls FOUT and then prints the contents of FOUT.BUF. You could now write a higher level routine, if you wish, which would examine the exponent to determine whether the number would fit in a 20-character field. If not, you could use QUICK.PRINT. If so, use FOUT with W=40 and D=18, and then truncate leading spaces and trailing zeroes. This would give you a complete print routine for any numbers, printing them in simple form when they fit and exponential form when they don't. Indeed, just such a routine already exists in DP18, but will have to wait for a future installment. FOUT can also be used as the base for a complete PRINT USING facility, and that is also already in DP18 waiting for future installments.

Meanwhile, enjoy these two conversions, and experiment with your own.

```
                  1000 *SAVE S.DP18 FOUT
                  1010 *--------------------------------
DB5C-             1020 AS.COUT        .EQ $DB5C
FFFF-             1030 DADD           .EQ $FFFF
FFFF-             1040 MOVE.YA.ARG    .EQ $FFFF
                  1050 *--------------------------------
0800-             1060 FOUT.BUF       .BS 41
29-               1070 FOUT.BUF.SIZE .EQ *-FOUT.BUF
                  1080 *--------------------------------
0829-             1090 W           .BS 1
082A-             1100 D           .BS 1
082B-             1110 INDEX       .BS 1
082C-             1120 SIGN.SIZE   .BS 1
082D-             1130 SIGN.CHAR   .BS 1
082E-             1140 ZERO.CHAR   .BS 1
082F-             1150 WW          .BS 1
0830-             1160 DD          .BS 1
0831-             1170 DIGIT.PICKER         .BS 1
0832-             1180 NO.LEADING.ZEROES    .BS 1
0833-             1190 NO.LEADING.DIGITS    .BS 1
0834-             1200 NO.INTEGRAL.ZEROES   .BS 1
0835-             1210 NO.LEADING.BLANKS    .BS 1
                  1220 *--------------------------------
0836-             1230 DAC            .BS 12
0836-             1240 DAC.EXPONENT   .EQ DAC
0837-             1250 DAC.HI         .EQ DAC+1
0840-             1260 DAC.EXTENSION  .EQ DAC+10
0841-             1270 DAC.SIGN       .EQ DAC+11
                  1280 *--------------------------------
```

```
0842-          ARG               .BS 12
0842-          ARG.EXPONENT      .EQ ARG
0843-          ARG.HI            .EQ ARG+1
084C-          ARG.EXTENSION     .EQ ARG+10
084D-          ARG.SIGN          .EQ ARG+11
               *--------------------------------
               *   QUICK PRINT
               *--------------------------------
               QUICK.PRINT
084E- 20 6A 08        JSR QUICK.FOUT
0851- 4C 5C 08        JMP FOR.PRINT.1
               *--------------------------------
               *   FORMATTED PRINT
               *       (A)=WIDTH OF FIELD
               *       (Y)=# OF FRACTIONAL DIGITS
               *--------------------------------
               FORMAT.PRINT
0854- A2 30           LDX #'0       USE ZEROES BEFORE FRACTION
0856- 8E 2E 08        STX ZERO.CHAR
0859- 20 F1 08        JSR FOUT
               *--------------------------------
               FOR.PRINT.1
085C- A0 00           LDY #0
085E- B9 00 08 .1     LDA FOUT.BUF,Y
0861- F0 06           BEQ .2
0863- 20 5C DB        JSR AS.COUT
0866- C8              INY
0867- D0 F5           BNE .1        ...ALWAYS
0869- 60       .2     RTS
               *--------------------------------
               *      QUICK CONVERSION
               *--------------------------------
               QUICK.FOUT
086A- A0 00           LDY #0
086C- 8C 2B 08        STY INDEX
086F- AD 36 08        LDA DAC.EXPONENT
0872- D0 0C           BNE .0        NUMBER IS NOT ZERO
0874- EE 2B 08        INC INDEX
0877- 8C 01 08        STY FOUT.BUF+1
087A- A9 30           LDA #'0
087C- 8D 00 08        STA FOUT.BUF  MAKE IT '0'
087F- 60              RTS
0880- AD 41 08 .0     LDA DAC.SIGN
0883- 10 05           BPL .1
0885- A9 2D           LDA #'-       NEGATIVE
0887- 20 01 0A        JSR STORE.CHAR
               *--------------------------------
088A- B9 37 08 .1     LDA DAC.HI,Y NEXT BYTE OF #
088D- 48              PHA
088E- 4A              LSR
088F- 4A              LSR
0890- 4A              LSR
0891- 4A              LSR
0892- 20 FD 09        JSR STORE.DIGIT
0895- C0 00           CPY #0
0897- D0 05           BNE .2
0899- A9 2E           LDA #'.       PUT DECIMAL POINT
089B- 20 01 0A        JSR STORE.CHAR
089E- 68       .2     PLA           DO 2ND DIGIT
089F- 20 FD 09        JSR STORE.DIGIT
08A2- C8              INY
08A3- C0 09           CPY #9        8 MORE BYTES
08A5- 90 E3           BCC .1
               *--------------------------------
08A7- AC 2B 08        LDY INDEX     TRUNCATE TRAILING ZEROS
08AA- 88       .3     DEY
08AB- B9 00 08        LDA FOUT.BUF,Y
08AE- C9 30           CMP #'0
08B0- F0 F8           BEQ .3        DONE
08B2- C9 2E           CMP #'.       TRAILING DECIMAL PT?
08B4- D0 01           BNE .4        NO
08B6- 88              DEY           YES, DELETE IT
08B7- C8       .4     INY
08B8- 8C 2B 08        STY INDEX     SAVE NEW END OF NUMBER
               *--------------------------------
08BB- A9 45           LDA #'E
08BD- 20 01 0A        JSR STORE.CHAR E FOR EXPONENT
08C0- A9 2B           LDA #'+
08C2- AC 36 08        LDY DAC.EXPONENT
```

```
08C5- 88                    DEY
08C6- C0 40                 CPY #$40
08C8- B0 02                 BCS .5
08CA- A9 2D                 LDA #'-
08CC- 20 01 0A      .5      JSR STORE.CHAR
08CF- 98                    TYA         EXPONENT
08D0- 38                    SEC
08D1- E9 40                 SBC #$40    REMOVE OFFSET
08D3- 10 04                 BPL .6
08D5- 49 FF                 EOR #$FF
08D7- 69 01                 ADC #1
08D9- A2 2F         .6      LDX #'0-1
08DB- 38                    SEC
08DC- E8            .8      INX
08DD- E9 0A                 SBC #10
08DF- B0 FB                 BCS .8
08E1- 69 3A                 ADC #'9+1
08E3- 48                    PHA
08E4- 8A                    TXA
08E5- 20 01 0A              JSR STORE.CHAR
08E8- 68                    PLA
08E9- 20 01 0A              JSR STORE.CHAR
08EC- A9 00                 LDA #0
08EE- 4C 01 0A              JMP STORE.CHAR
                    *--------------------------------
                    *   FORMATTED CONVERSION
                    *       (A)=WIDTH OF FIELD
                    *       (Y)=# OF FRACTIONAL DIGITS
                    *--------------------------------
                    FOUT
08F1- C9 29                 CMP #FOUT.BUF.SIZE    LIMIT WIDTH
08F3- 90 02                 BCC .1
08F5- A9 28                 LDA #FOUT.BUF.SIZE-1
08F7- 8D 29 08      .1      STA W
08FA- CC 29 08              CPY W       FORCE D<W
08FD- 90 02                 BCC .2
08FF- A8                    TAY
0900- 88                    DEY
0901- 8C 2A 08      .2      STY D
                    *--------------------------------
0904- A9 00                 LDA #0
0906- 8D 2B 08              STA INDEX
0909- 8D 2C 08              STA SIGN.SIZE
090C- 8D 2D 08              STA SIGN.CHAR
090F- 8D 34 08              STA NO.INTEGRAL.ZEROES
0912- 8D 32 08              STA NO.LEADING.ZEROES
0915- 8D 31 08              STA DIGIT.PICKER
                    *--------------------------------
0918- 20 0B 0A              JSR ROUND.DAC.D   ROUND TO D DIGITS
091B- AD 41 08              LDA DAC.SIGN
091E- 10 08                 BPL .3
0920- EE 2C 08              INC SIGN.SIZE
0923- A9 2D                 LDA #'-     MINUS SIGN
0925- 8D 2D 08              STA SIGN.CHAR
                    *--------------------------------
0928- 38            .3      SEC
0929- AD 36 08              LDA DAC.EXPONENT
092C- E9 40                 SBC #$40    REMOVE OFFSET
092E- 10 0A                 BPL .4
0930- 49 FF                 EOR #$FF
0932- 8D 32 08              STA NO.LEADING.ZEROES
0935- EE 32 08              INC NO.LEADING.ZEROES
0938- A9 00                 LDA #0
093A- 8D 33 08      .4      STA NO.LEADING.DIGITS
                    *--------------------------------
093D- 38                    SEC
093E- AD 33 08              LDA NO.LEADING.DIGITS
0941- E9 12                 SBC #18
0943- 30 08                 BMI .5
0945- 8D 34 08              STA NO.INTEGRAL.ZEROES
0948- A9 12                 LDA #18     18 SIGNIF DIGITS MAX
094A- 8D 33 08              STA NO.LEADING.DIGITS
                    *--------------------------------
094D- 18            .5      CLC         CALCULATE TOTAL # OF DIGITS
094E- AD 2C 08              LDA SIGN.SIZE
0951- 6D 33 08              ADC NO.LEADING.DIGITS
0954- 6D 34 08              ADC NO.INTEGRAL.ZEROES
0957- 6D 2A 08              ADC D
095A- 69 01                 ADC #1
```

```
095C- 8D 2F 08          STA WW
                 *-------------------------------
095F- 38                SEC
0960- AD 2A 08          LDA D
0963- ED 32 08          SBC NO.LEADING.ZEROES
0966- 8D 30 08          STA DD
                 *-------------------------------
0969- 38                SEC
096A- AD 29 08          LDA W
096D- ED 2F 08          SBC WW
0970- 30 59             BMI .14       ...OVERFLOW
0972- 8D 35 08          STA NO.LEADING.BLANKS
0975- AD 33 08          LDA NO.LEADING.DIGITS
0978- D0 08             BNE .6
097A- CE 35 08          DEC NO.LEADING.BLANKS
097D- 10 03             BPL .6
097F- EE 35 08          INC NO.LEADING.BLANKS IT WENT -, MAKE 0
                 *---STORE LEADING BLANKS---------
0982- A9 20      .6     LDA #' '      BLANK
0984- AC 35 08          LDY NO.LEADING.BLANKS
0987- 20 FA 09          JSR STORE.N.CHARS
                 *---STORE SIGN-------------------
098A- AD 2D 08          LDA SIGN.CHAR
098D- F0 03             BEQ .8
098F- 20 01 0A          JSR STORE.CHAR
                 *---STORE INTEGRAL DIGITS--------
0992- AC 33 08   .8     LDY NO.LEADING.DIGITS
0995- F0 05             BEQ .10
0997- 20 F3 09          JSR STORE.N.DIGITS
099A- F0 06             BEQ .11       ...ALWAYS
099C- AD 2E 08   .10    LDA ZERO.CHAR  NO INTEGER PART,SO PRINT 0
099F- 20 01 0A          JSR STORE.CHAR
09A2- A9 30      .11    LDA #'0
09A4- AC 34 08          LDY NO.INTEGRAL.ZEROES
09A7- 20 FA 09          JSR STORE.N.CHARS
                 *---STORE FRACTION---------------
09AA- A9 2E             LDA #'.
09AC- 20 01 0A          JSR STORE.CHAR
09AF- AD 30 08          LDA DD
09B2- 0D 32 08          ORA NO.LEADING.ZEROES
09B5- F0 0F             BEQ .13
09B7- AD 2E 08          LDA ZERO.CHAR
09BA- AC 32 08          LDY NO.LEADING.ZEROES
09BD- 20 FA 09          JSR STORE.N.CHARS
09C0- AC 30 08          LDY DD
09C3- 20 F3 09          JSR STORE.N.DIGITS
                 *---TERMINATE STRING-------------
09C6- A9 00      .13    LDA #0
09C8- 4C 01 0A          JMP STORE.CHAR
                 *-------------------------------
09CB- A9 2A      .14    LDA #'*'      FILL FIELD WITH STARS
09CD- AC 29 08          LDY W
09D0- 20 FA 09          JSR STORE.N.CHARS
09D3- 4C C6 09          JMP .13
                 *-------------------------------
                 *   STORE NEXT (Y) DIGITS
                 *-------------------------------
09D6- AD 31 08   SND..1 LDA DIGIT.PICKER
09D9- C9 14             CMP #20
09DB- 90 04             BCC .1
09DD- A9 00             LDA #0
09DF- F0 0E             BEQ .2        ...ALWAYS
09E1- 4A         .1     LSR           LEFT/RIGHT --> C
09E2- AA                TAX           INDEX --> X
09E3- EE 31 08          INC DIGIT.PICKER
09E6- BD 37 08          LDA DAC.HI,X
09E9- B0 04             BCS .2
09EB- 4A                LSR
09EC- 4A                LSR
09ED- 4A                LSR
09EE- 4A                LSR
09EF- 20 FD 09   .2     JSR STORE.DIGIT
09F2- 88                DEY
                 STORE.N.DIGITS
09F3- D0 E1             BNE SND..1
09F5- 60                RTS
                 *-------------------------------
```

Enable/Disable IRQ from Applesoft..........Bob Sander-Cederlof

If you have applied the patches to DOS 3.3 that we published in the January 1984 issue (pages 10,11), and if you now are using interrupts from such sources as the Timemaster II or a handy pushbutton, you have probably run into the need to enable and disable IRQ from within an Applesoft program. (That sentence is the kind you have to read without interruption, so I really should have begun the paragraph with SEI and ended it with CLI.)

What is need is four bytes of assembly language, at a location that you can CALL. For example:

```
300- 58    CLI
301- 60    RTS
302- 78    SEI
303- 60    RTS
```

If those four bytes are in memory as shown, you can CALL 768 to enable IRQ interrupts, and CALL 771 to disable them. You can install the four bytes like this:

```
100 POKE 768,88: POKE 769,96
110 POKE 770,120:POKE 771,96
```

Now there are often times when poking into page 3 is not possible. Are there other tricky ways to get those bytes installed, without using page 3?

I found a half dozen or so. First, realize that the four bytes only need to be there when you call them. The rest of the time the same locations could be used for other purposes. For example, we could poke them into the input buffer at $200, as long as we do it every time we CALL it:

```
100 POKE 512,88:POKE 513,96:CALL 512
      to enable interrupts, or

500 POKE 512,120:POKE 513,96:CALL 512
      to disable them.
```

The result of a multiplication or division is left, sometimes normalized and sometimes not, in $62...$66. If we find two numbers whose product leaves the bytes $58 and $60 at $62 and $63, we could CALL 98:

```
100 X = 1*707 : CALL 98 : REM ENABLE IRQ
200 X = 1*963 : CALL 98 : REM DISABLE IRQ
```

On the next page is a table showing the various methods I found:

```
Enable (CLI..RTS)         Disable (SEI..RTS)
-------------------------------------------------------
100 POKE 38,88            100 POKE 38,120
110 POKE 39,96            110 POKE 39,96
120 CALL 38               120 CALL 38
-------------------------------------------------------
100 CALL 8411232-8411065  100 CALL 8419424-8419257
-------------------------------------------------------
100 GOSUB 24664           100 GOSUB 24696
...                       ...
24664 CALL 117:RETURN     24696 CALL 117:RETURN
-------------------------------------------------------
100 X = 1*707 : CALL 98   100 X = 1*963 : CALL 98
-------------------------------------------------------
100 X = RND(-8411323.5)   100 X = RND(-8419424.5)
110 CALL 203              110 CALL 203
-------------------------------------------------------
100 HOME:FLASH:PRINT"X "  100 HOME:FLASH:PRINT"8 "
110 NORMAL:CALL1024       110 NORMAL:CALL1024
-------------------------------------------------------
```

Can you figure out how all these work? They are pretty tricky! Can you think of some more?

Line Number Cross Reference...........................Bill Morgan

Have you ever had to modify a BASIC program written by someone who didn't seem to know what he was doing? Deciphering several hundred undocumented lines of split FOR/NEXTs and tangled GOTOs can lead to a severe headache. We recently had a consulting job that involved just such a project: one program to be altered was about a hundred sectors of spaghetti-plate Applesoft. A couple of the biggest problems were figuring out which lines used a particular variable, and what lines called others, or were called from where.

Back in November of 1980, AAL published a Variable Cross Reference program which neatly took care of the first problem by producing a listing in alphabetical order of all the variables used and all the lines using them. At the end of that article, Bob S-C pointed out that the program could, with some effort, be modified into just the sort of Line Number Cross Reference we now needed. Well, I drew the job of making that modification, and here's what I came up with.

The Basis

These Cross Reference programs use a hash-chain data structure to store the called and calling line numbers. Each called line has its own list of lines which refer to it. We locate these lists by using the upper six bits of the line number for an index into a table located at $280. This table contains the address of the beginning of each of the 64 possible chains. Each chain is made up of the data for a range of 1024 possible called line numbers. The first one has called lines 0-1023, the second has 1024-2047, and so on.

The entry for each called line is made up of a pointer to the next called line in that chain, this called line number, a pointer to the next calling line, and the number of this calling line. Each subsequent calling line entry has only the last four bytes. A pointer with a value of zero marks the end of each chain and each list.

VCR used three characters for each variable: the first two letters of the variable name and a type designator of "$", "%" or " ". The first character was the hash index and the last two characters were stored at the beginning of each variable's chain. LCR uses the high-order 6 bits of the called line number for the hash index and stores both bytes of the number in the chain. This is slightly redundant, so if you want to store more information about the called line, you can use the upper six bits of the chain entry.

VCR stored the calling line numbers with the high byte first, backwards from usual 6502 practice. This was done so the same search-compare code could handle both variable names and line numbers. To simplify the conversion I kept the same structure, even though it's no longer strictly necessary.

The Program

LCR, the overall control level, is identical to VCR and just calls the other routines.

INITIALIZATION prepares a couple of pointers and zeroes the hash table. The only difference here is the size of the hash table.

PROCESS.LINE is also the same as in VCR. This routine steps through the lines of the Applesoft program, moving the calling line number into our data area and JSRing to SCAN.FOR.CALLS to work on each line.

SCAN.FOR.CALLS is the first really new section of code. We start by setting a flag used to mark ON ... GO statements. Then we step through the bytes of the line, looking for tokens that call another line. GOTO and GOSUB are processed immediately. For a THEN token we check to see if the next character is a number. If it is, we deal with it; if not, we go on. If we find an ON token, we set the flag and keep looking. After a GOTO or GOSUB we check ONFLAG. If there was an ON, we look for a comma to mark another called line number.

PROCESS.CALL first converts the ASCII line number of the called line into a two-byte binary number and then searches the data structure for that line. If it is there, we simply add this calling line to the list. If we don't find the called line we create a new entry for it.

CONVERT.LINE.NUMBER is lifted straight from Applesoft's LINGET, at $DA0C.

NEXT.CHAR is a utility routine to get the next byte from the program and advance the pointer.

SEARCH.CALL.TABLE starts the search pointer on the appropriate chain.

CHAIN.SEARCH uses the pointer in an entry to step to the next entry. If the pointer is zero, then there is no next entry and the search has failed. We then compare the line number in the entry to the one we're looking for. If the entry is less than the search key, we go on. If it is equal, we update the pointer and report success. If we hit an entry greater than the key, the search fails and we return.

SEARCH.LINE.CHAIN is called after SEARCH.CALL.TABLE has found a match. Here we move the pointer to the calling line field of the matching entry and use the current calling line for a search key.

ADD.NEW.ENTRY first updates the pointers in the previous entry and this new entry, and the end-of-table pointer. We then make sure there is room for the new entry and move the data up into the new space.

```
                  3640 *   STORE (Y) OF THE CHARACTER IN (A)
                  3650 *       (Z-STATUS IF COUNT IS 0)
                  3660 *--------------------------------
09F6- 20 01 0A    3670 SNC..1 JSR STORE.CHAR
09F9- 88          3680        DEY
                  3690 STORE.N.CHARS
09FA- D0 FA       3700        BNE SNC..1
09FC- 60          3710        RTS
                  3720 *--------------------------------
                  3730 *      STORE A CHAR IN THE BUFFER
                  3740 *--------------------------------
                  3750 STORE.DIGIT
09FD- 29 0F       3760        AND #$0F
09FF- 09 30       3770        ORA #'0'
                  3780 STORE.CHAR
0A01- AE 2B 08    3790        LDX INDEX
0A04- 9D 00 08    3800        STA FOUT.BUF,X
0A07- EE 2B 08    3810        INC INDEX
0A0A- 60          3820        RTS
                  3830 *--------------------------------
                  3840 *      ROUND DAC TO (D) DECIMAL PLACES
                  3850 *--------------------------------
                  3860 ROUND.DAC.D
0A0B- AD 41 08    3870        LDA DAC.SIGN  GET THE SIGN
0A0E- 48          3880        PHA           SAVE IT
0A0F- A9 28       3890        LDA #CON.1HALF
0A11- A0 0A       3900        LDY /CON.1HALF
0A13- 20 FF FF    3910        JSR MOVE.YA.ARG   MOVE .5*10^-D INTO ARG
0A16- 68          3920        PLA           GET SIGN
0A17- 8D 4D 08    3930        STA ARG.SIGN
0A1A- AD 2A 08    3940        LDA D         GET # OF PLACES
0A1D- 49 FF       3950        EOR #$FF      MAKE IT NEGATIVE BY 2S COMPLEMENT
0A1F- 38          3960        SEC           ADD 1 DURING NEXT ADD
0A20- 69 40       3970        ADC #$40      ADD OFFSET
0A22- 8D 42 08    3980        STA ARG.EXPONENT
0A25- 4C FF FF    3990        JMP DADD      ADD .5*10^-D;FOUT WILL TRUNCATE IT
                  4000 *--------------------------------
0A28- 40 50 00
0A2B- 00 00 00
0A2E- 00 00 00
0A31- 00 00       4010 CON.1HALF .HS 40500000000000000000
```

27 Enhancements for the S-C Macro Assembler 1.1 (Videx $1000 ver.)

A 1723-byte patch for Videx 80-column board users providing On-Line editing with version 1.1 - All the edit commands are now available during normal input (non-edit mode) i.e. insert, delete, skip-to-character etc. The New/Extended Commands are:

<u>Non-Edit Mode:</u> Ctrl-A,B,C,D,Fx,K,L,M,N,P,Q,R,S,T,V,W,Z,@,
Ctrl-shift-N, Esc-C, PAG, USR, AUTO, VAL, MEM.
<u>Edit Mode:</u> Ctrl-K, P, Z.

The RETURN-key accepts the whole line irrespective of cursor position. USR toggles line-numbers on/off during listing and assembly. VAL now displays hex, binary and decimal results. Ctrl-K skips to start of comment column from any position. Ctrl-Z does an UPPER/lower-case toggle. Ctrl-W toggles between AUTO and MANUAL modes. After a CTRL-shift-N the next key pressed becomes the clear-to-tab character. Esc-C does a CATALOG. MEM shows free space in RAM and on Disk. (DOS 3.3) Ctrl-C continues editing from current linenumber. Ctrl-V allows the user to view source from current linenumber. Leading zeros are removed from linenumbers and page numbers if using .TI. The SYM routine is modified to exclude the clutter of local labels. Ctrl-P outputs the source separator "*-----etc" in both the edit and non-edit modes. You can now use Ver 1.1 as a useful if limited 80-column line-oriented wordprocessor for your letters and reports. User RAM $3C00-$9600 (Maxfiles 3). All for $15 - which buys a disk with Article, binary patch, installer and 980-line source code.
R.F.O'Brien,14 CLonshaugh Lawn, Coolock, Dublin 5, Ireland.

Now we are done with the routines devoted to building the Cross Reference tables. Interestingly, SEARCH.CALL.TABLE, CHAIN.SEARCH, SEARCH.LINE.CHAIN, and ADD.NEW.ENTRY are the real heart of this program, and the only change I had to make in these routines from VCR to LCR was to alter the method of figuring the hash index in SEARCH.CALL.TABLE. Next we come to getting the data back out of the tables and onto a display.

PRINT.REPORT first sets a pointer we'll be using later on and then steps through the hash table, calling PRINT.CHAIN for each entry found.

PRINT.CHAIN starts out by checking for a pause or abort signal from the keyboard. It then moves the current called line number into LINNUM, checks to see if it really exists, and prints it, followed by an asterisk if it is undefined. Now we move a pointer up to the start of the calling line list and call PRINT.LINNUM.CHAIN to display all the entries. The last step is to move the pointer up to the next called line in this chain, if any, and go back to do that one.

CHECK.DEFINITION keeps its own pointer into the program and steps along checking each called line to see if it actually exists. It provides a space or an asterisk to be printed after the line number.

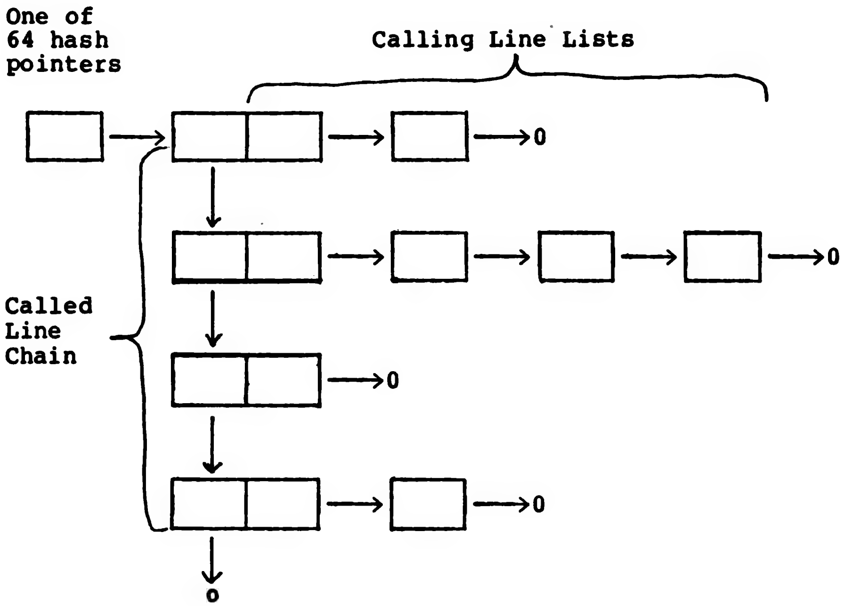
PRINT.LINNUM.CHAIN displays the calling lines stored for each called line. We first tab to the next column (or line if necessary), then get the line number out of the list and print it. Lastly, we move the pointer up to the next entry, if any, and loop back.

TAB.NEXT.COLUMN prints enough blanks to move over to the next output position. If a new line is necessary, it checks the line number to see if the new line should go to the screen only, or also to a printer. This is Louis Pitz's addition, designed to automatically handle either 40- or 80-column output.

PRINT.LINE.NUMBER and CHECK.FOR.PAUSE are pretty standard routines to convert a two-byte binary number into five decimal characters, and to provide for pause/abort during display.

Well, now we have a Line Number Cross Reference to go along with the Variable Cross Reference. Now all that remains is to integrate the two programs into one master Applesoft Cross Reference Utility. Maybe you could call it with "&V" for VCR, or "&L" for LCR, and simply "&" to get both listings. Any takers out there?

PS: Bob suggested that I add a diagram of the hash chain structure, and a summary of the search process. OK, here they are...



Found a Call

* Use high 6 bits of called line number to index Hash Table
* Get pointer from Hash Table to find start of chain
    * If no pointer in Hash Table, make new entry
* Search chain for same line number
    * If not found, make new link in chain
    * If found, search calling line list
* Enter new calling line in list

```
                     1000 *SAVE S.LCR
                     1010 *--------------------------------
                     1020 *     LINE NUMBER CROSS REFERENCE
                     1030 *       FOR APPLESOFT PROGRAMS
                     1040 *
                     1050 *       Based on Variable Cross Reference
                     1060 *       Original by Bob S-C 11/80
                     1070 *       Modified by Louis Pitz 8/83
                     1080 *       Adapted by Bill Morgan 8/84
                     1090 *--------------------------------
                     1100        .OR $9000
                     1110        .TF B.LCR
                     1120 *--------------------------------
9000- A9 4C          1130        LDA #$4C      set & vector
9002- 8D F5 03       1140        STA $3F5
9005- A9 10          1150        LDA #LCR
9007- 8D F6 03       1160        STA $3F6
900A- A9 90          1170        LDA /LCR
900C- 8D F7 03       1180        STA $3F7
900F- 60             1190        RTS
                     1200 *--------------------------------
15-                  1210 TEMP    .EQ $15
16-                  1220 COUNTER .EQ $16
17-                  1230 ONFLAG  .EQ $17              ON ... GO flag
17-                  1240 DEFFLAG .EQ $17
18-                  1250 PNTR    .EQ $18,19           pointer into program
1A-                  1260 LZFLAG  .EQ $1A              leading zero flag
```

```
1A-                 1270 DATA     .EQ $1A thru $1D
1A-                 1280 NEXTLN   .EQ $1A,1B        address of next line
1C-                 1290 LINNUM   .EQ $1C,1D        current line number
1E-                 1300 STPNTR   .EQ $1E,1F        pointer into call table
9B-                 1310 TPTR     .EQ $9B,9C        temp pointer
9D-                 1320 ENTRY    .EQ $9D thru $A4  8 bytes
9F-                 1330 CALL     .EQ ENTRY+2
A5-                 1340 SIZE     .EQ $A5,A6
0280-               1350 HSHTBL   .EQ $280
                    1360 *--------------------------------
67-                 1370 PRGBOT   .EQ $67,68        beginning of program
69-                 1380 LOMEM    .EQ $69,6A        beginning of variable space
6B-                 1390 EOT      .EQ $6B,6C        end of variable table
                    1400 *--------------------------------
2C-                 1410 COMMA        .EQ ',
8D-                 1420 CR           .EQ $8D
AB-                 1430 TKN.GOTO     .EQ $AB
B0-                 1440 TKN.GOSUB    .EQ $B0
B4-                 1450 TKN.ON       .EQ $B4
C4-                 1460 TKN.THEN     .EQ $C4
                    1470 *--------------------------------
24-                 1480 MON.CH       .EQ $24
C000-               1490 KEYBOARD     .EQ $C000
C010-               1500 STROBE       .EQ $C010
D410-               1510 AS.MEMFULL   .EQ $D410
F94A-               1520 MON.PRBL2    .EQ $F94A
FD8E-               1530 MON.CROUT    .EQ $FD8E
FDED-               1540 MON.COUT     .EQ $FDED
FDF0-               1550 MON.COUT1    .EQ $FDF0
                    1560 *--------------------------------
9010- 20 23 90      1570 LCR      JSR INITIALIZATION
9013- 20 3E 90      1580 .1       JSR PROCESS.LINE
9016- D0 FB         1590          BNE .1             until end of program
9018- 20 B5 91      1600          JSR PRINT.REPORT
901B- 20 23 90      1610          JSR INITIALIZATION  erase call table
901E- A9 00         1620          LDA #0             clear $A4 so Applesoft
9020- 85 A4         1630          STA $A4            will work correctly
9022- 60            1640          RTS
                    1650 *--------------------------------
                    1660 INITIALIZATION
9023- A5 69         1670          LDA LOMEM          start call table
9025- 85 6B         1680          STA EOT            after program
9027- A5 6A         1690          LDA LOMEM+1
9029- 85 6C         1700          STA EOT+1
902B- A2 80         1710          LDX #$80           # of bytes for hash pointers
902D- A9 00         1720          LDA #0
902F- 9D 7F 02      1730 .1       STA HSHTBL-1,X
9032- CA            1740          DEX
9033- D0 FA         1750          BNE .1
9035- A5 67         1760          LDA PRGBOT         start pointer at
9037- 85 18         1770          STA PNTR           beginning of program
9039- A5 68         1780          LDA PRGBOT+1
903B- 85 19         1790          STA PNTR+1
903D- 60            1800          RTS
                    1810 *--------------------------------
                    1820 PROCESS.LINE
903E- A0 03         1830          LDY #3             capture pointer and line #
9040- B1 18         1840 .1       LDA (PNTR),Y
9042- 99 1A 00      1850          STA DATA,Y
9045- 88            1860          DEY
9046- 10 F8         1870          BPL .1
9048- A5 1B         1880          LDA DATA+1         check if end
904A- F0 16         1890          BEQ .3             yes, return .EQ.
904C- 18            1900          CLC
904D- A5 18         1910          LDA PNTR           adjust pointer to
904F- 69 04         1920          ADC #4             skip over data
9051- 85 18         1930          STA PNTR
9053- 90 02         1940          BCC .2
9055- E6 19         1950          INC PNTR+1
9057- 20 63 90      1960 .2       JSR SCAN.FOR.CALLS
905A- A5 1A         1970          LDA DATA           point to next line
905C- 85 18         1980          STA PNTR
905E- A5 1B         1990          LDA DATA+1         and return .NE.
9060- 85 19         2000          STA PNTR+1
9062- 60            2010 .3       RTS
                    2020 *--------------------------------
```

```
                     SCAN.FOR.CALLS
9063- A9 FF                 LDA #$FF
9065- 85 17                 STA ONFLAG
9067- 20 07 91       .1     JSR NEXT.CHAR
906A- F0 32                 BEQ .4
906C- C9 C4                 CMP #TKN.THEN     scan for call token
906E- F0 10                 BEQ .2
9070- C9 AB                 CMP #TKN.GOTO
9072- F0 18                 BEQ .3
9074- C9 B0                 CMP #TKN.GOSUB
9076- F0 14                 BEQ .3
9078- C9 B4                 CMP #TKN.ON
907A- D0 EB                 BNE .1            no match, keep going
907C- 46 17                 LSR ONFLAG        set flag for ON token
907E- 10 E7                 BPL .1         ...always

9080- A0 00          .2     LDY #0            after THEN, check
9082- B1 18                 LDA (PNTR),Y      for line number
9084- C9 30                 CMP #'0
9086- 90 DF                 BCC .1            <0 isn't
9088- C9 3A                 CMP #'9+1
908A- B0 DB                 BCS .1            neither is >9

908C- 20 9F 90       .3     JSR PROCESS.CALL  handle this call
908F- A5 17                 LDA ONFLAG        are we in ON?
9091- 30 D0                 BMI SCAN.FOR.CALLS no, go on
9093- 20 07 91              JSR NEXT.CHAR     yes, look for comma
9096- F0 06                 BEQ .4            EOL
9098- C9 2C                 CMP #COMMA
909A- F0 F0                 BEQ .3            comma, get another call
909C- D0 C5                 BNE SCAN.FOR.CALLS ...always

909E- 60             .4     RTS
                     *--------------------------------
                     PROCESS.CALL
909F- 20 C4 90              JSR CONVERT.LINE.NUMBER
90A2- 20 14 91              JSR SEARCH.CALL.TABLE
90A5- 90 13                 BCC .2            found same call
90A7- A9 00                 LDA #0
90A9- 85 A1                 STA ENTRY+4       start of line number chain
90AB- 85 A2                 STA ENTRY+5
90AD- A5 1D                 LDA LINNUM+1      MSB first
90AF- 85 A3                 STA ENTRY+6
90B1- A5 1C                 LDA LINNUM
90B3- 85 A4                 STA ENTRY+7
90B5- A9 08                 LDA #8            add 8 byte entry
90B7- 4C 74 91       .1     JMP ADD.NEW.ENTRY

90BA- 20 54 91       .2     JSR SEARCH.LINE.CHAIN
90BD- 90 04                 BCC .3            found same line number
90BF- A9 04                 LDA #4            add 4 byte entry
90C1- D0 F4                 BNE .1         ...always

90C3- 60             .3     RTS
                     *--------------------------------
                     CONVERT.LINE.NUMBER
90C4- A9 00                 LDA #0
90C6- 85 A0                 STA CALL+1
90C8- 85 9F                 STA CALL
90CA- 20 07 91       .1     JSR NEXT.CHAR
90CD- F0 2F                 BEQ .2            EOL
90CF- 38                    SEC
90D0- E9 30                 SBC #'0           make value
90D2- 90 2A                 BCC .2            <0 isn't number
90D4- C9 0A                 CMP #9+1
90D6- B0 26                 BCS .2            >9 isn't number
90D8- 48                    PHA               save value
90D9- A5 9F                 LDA CALL
90DB- 85 15                 STA TEMP
90DD- A5 A0                 LDA CALL+1        multiply CALL * 10
90DF- 0A                    ASL
90E0- 26 15                 ROL TEMP
90E2- 0A                    ASL
90E3- 26 15                 ROL TEMP
90E5- 65 A0                 ADC CALL+1
90E7- 85 A0                 STA CALL+1
90E9- A5 15                 LDA TEMP
90EB- 65 9F                 ADC CALL
90ED- 85 9F                 STA CALL
```

```
90EF- 06 A0              ASL CALL+1
90F1- 26 9F              ROL CALL
90F3- 68                 PLA              get value this digit
90F4- 65 A0              ADC CALL+1       and add it in
90F6- 85 A0              STA CALL+1
90F8- 90 D0              BCC .1
90FA- E6 9F              INC CALL
90FC- B0 CC              BCS .1       ...always

90FE- A5 18     .2       LDA PNTR         back up PNTR
9100- D0 02              BNE .3
9102- C6 19              DEC PNTR+1
9104- C6 18     .3       DEC PNTR
9106- 60                 RTS
                *--------------------------------
                NEXT.CHAR
9107- A0 00              LDY #0
9109- B1 18              LDA (PNTR),Y
910B- F0 06              BEQ .1           EOL
910D- E6 18              INC PNTR         bump pointer
910F- D0 02              BNE .1
9111- E6 19              INC PNTR+1
9113- 60        .1       RTS
                *--------------------------------
                SEARCH.CALL.TABLE
9114- A5 9F              LDA CALL         hi-byte of called line
9116- 29 FC              AND #$FC         hi 6 bits
9118- 4A                 LSR              make 0-126
9119- 69 80              ADC #HSHTBL      carry is clear
911B- 85 1E              STA STPNTR
911D- A9 02              LDA /HSHTBL
911F- 69 00              ADC #0
9121- 85 1F              STA STPNTR+1
                *--- fall into CHAIN.SEARCH routine
                *--------------------------------
                CHAIN.SEARCH
9123- A0 00     .1       LDY #0           point at pointer in entry
9125- B1 1E              LDA (STPNTR),Y
9127- 85 9B              STA TPTR
9129- C8                 INY
912A- B1 1E              LDA (STPNTR),Y
912C- F0 1A              BEQ .4           end of chain, not in table
912E- 85 9C              STA TPTR+1
9130- A2 02              LDX #2           2 bytes in number
9132- A0 02              LDY #2           point at line number in entry
9134- B1 9B     .2       LDA (TPTR),Y     compare numbers
9136- D9 9D 00           CMP ENTRY,Y
9139- 90 08              BCC .3           not this one, but keep looking
913B- D0 0B              BNE .4           not in this chain
913D- CA                 DEX
913E- F0 0A              BEQ .5           same number
9140- C8                 INY              next byte pair
9141- D0 F1              BNE .2       ...always

9143- 20 4A 91  .3       JSR .5           update pointer, clear carry
9146- 90 DB              BCC .1       ...always

9148- 38        .4       SEC              did not find
9149- 60                 RTS

914A- A5 9B     .5       LDA TPTR         point to matching entry
914C- 85 1E              STA STPNTR
914E- A5 9C              LDA TPTR+1
9150- 85 1F              STA STPNTR+1
9152- 18                 CLC
9153- 60                 RTS
                *--------------------------------
                SEARCH.LINE.CHAIN
9154- 18                 CLC              adjust pointer to start
9155- A5 1E              LDA STPNTR       of line # chain
9157- 69 04              ADC #4
9159- 85 9D              STA ENTRY
915B- A5 1F              LDA STPNTR+1
915D- 69 00              ADC #0
915F- 85 9E              STA ENTRY+1
9161- A9 9D              LDA #ENTRY
9163- 85 1E              STA STPNTR
9165- A9 00              LDA /ENTRY
9167- 85 1F              STA STPNTR+1
```

```
9169- A5 1C             LDA LINNUM       put line number into symbol
916B- 85 A0             STA ENTRY+3
916D- A5 1D             LDA LINNUM+1
916F- 85 9F             STA ENTRY+2
9171- 4C 23 91          JMP CHAIN.SEARCH
                  *--------------------------------
                  ADD.NEW.ENTRY
9174- 85 A5             STA SIZE
9176- 18                CLC              see if room
9177- A2 01             LDX #1
9179- A0 00             LDY #0
917B- 84 A6             STY SIZE+1
917D- B1 1E       .1    LDA (STPNTR),Y   get current pointer
917F- 99 9D 00          STA ENTRY,Y      into new entry
9182- B9 6B 00          LDA EOT,Y        point old entry
9185- 91 1E             STA (STPNTR),Y   to this one
9187- 99 9B 00          STA TPTR,Y
918A- 79 A5 00          ADC SIZE,Y       and adjust end-of-table
918D- 99 6B 00          STA EOT,Y
9190- C8                INY
9191- CA                DEX
9192- 10 E9             BPL .1           now do low-bytes
                  *--- see if there's going to be enough room
9194- A5 6B             LDA EOT
9196- C9 10             CMP #LCR
9198- A5 6C             LDA EOT+1
919A- E9 90             SBC /LCR
919C- B0 14             BCS .3           MEM FULL error
                  *--- move entry into call table
919E- A4 A5             LDY SIZE
91A0- 88                DEY
91A1- B9 9D 00    .2    LDA ENTRY,Y
91A4- 91 9B             STA (TPTR),Y
91A6- 88                DEY
91A7- 10 F8             BPL .2
91A9- A5 9B             LDA TPTR
91AB- 85 1E             STA STPNTR
91AD- A5 9C             LDA TPTR+1
91AF- 85 1F             STA STPNTR+1
91B1- 60                RTS

91B2- 4C 10 D4    .3    JMP AS.MEMFULL   abort with error message
                  *--------------------------------
                  PRINT.REPORT
91B5- A5 67             LDA PRGBOT
91B7- 85 18             STA PNTR         start defined line search
91B9- A5 68             LDA PRGBOT+1     at beginning of program
91BB- 85 19             STA PNTR+1
91BD- A9 00             LDA #0           start at chain 0
91BF- 85 15       .1    STA TEMP
91C1- 0A                ASL
91C2- A8                TAY
91C3- B9 81 02          LDA HSHTBL+1,Y
91C6- F0 0A             BEQ .2           no entries for this chain
91C8- 85 1F             STA STPNTR+1
91CA- B9 80 02          LDA HSHTBL,Y
91CD- 85 1E             STA STPNTR
91CF- 20 DB 91          JSR PRINT.CHAIN
91D2- E6 15       .2    INC TEMP
91D4- A5 15             LDA TEMP
91D6- C9 40             CMP #$40
91D8- 90 E5             BCC .1           still more chains
91DA- 60                RTS              finished
                  *--------------------------------
                  PRINT.CHAIN
91DB- 20 DC 92          JSR CHECK.FOR.PAUSE
91DE- F0 3A             BEQ .1           <CR> abort
91E0- A0 02             LDY #2
91E2- B1 1E             LDA (STPNTR),Y
91E4- 85 1D             STA LINNUM+1
91E6- C8                INY
91E7- B1 1E             LDA (STPNTR),Y
91E9- 85 1C             STA LINNUM
91EB- 20 1D 92          JSR CHECK.DEFINITION
91EE- 20 95 92          JSR PRINT.LINE.NUMBER
91F1- A5 17             LDA DEFFLAG      "*" or " "
91F3- 20 ED FD          JSR MON.COUT
```

```
91F6- 18              CLC
91F7- A5 1E           LDA STPNTR
91F9- 69 04           ADC #4             point at line # chain
91FB- 85 9B           STA TPTR
91FD- A5 1F           LDA STPNTR+1
91FF- 69 00           ADC #0
9201- 85 9C           STA TPTR+1
9203- 20 46 92        JSR PRINT.LINNUM.CHAIN
9206- 20 8E FD        JSR MON.CROUT
9209- A0 01           LDY #1
920B- B1 1E           LDA (STPNTR),Y     pointer to next call
920D- F0 0D           BEQ .2             no more
920F- 48              PHA
9210- 88              DEY
9211- B1 1E           LDA (STPNTR),Y
9213- 85 1E           STA STPNTR
9215- 68              PLA
9216- 85 1F           STA STPNTR+1
9218- D0 00           BNE .1          ...always

921A- 68         .1   PLA                return to top level
921B- 68              PLA                if <CR> abort
921C- 60         .2   RTS
                 *-------------------------------
                 CHECK.DEFINITION
921D- A0 03           LDY #3
921F- A2 01           LDX #1
9221- B1 18      .1   LDA (PNTR),Y       look at next line in program
9223- D5 1C           CMP LINNUM,X
9225- 90 0F           BCC .4             < our number, get new line
9227- D0 08           BNE .2             >   "      "  , not defined
9229- 88              DEY                =   "      "  , go on
922A- CA              DEX                now do low order bytes
922B- 10 F4           BPL .1
922D- A9 A0           LDA #" "           found it!
922F- D0 02           BNE .3          ...always

9231- A9 AA      .2   LDA #"*"           flag undefined line
9233- 85 17      .3   STA DEFFLAG
9235- 60              RTS

9236- A0 00      .4   LDY #0
9238- B1 18           LDA (PNTR),Y       lo-byte of next line address
923A- 48              PHA
923B- C8              INY
923C- B1 18           LDA (PNTR),Y       and hi-byte
923E- 85 19           STA PNTR+1
9240- 68              PLA
9241- 85 18           STA PNTR
9243- 4C 1D 92        JMP CHECK.DEFINITION
                 *-------------------------------
                 PRINT.LINNUM.CHAIN
9246- A9 00           LDA #0             reset counter to 0
9248- 85 16           STA COUNTER        for each call
924A- 20 70 92   .1   JSR TAB.NEXT.COLUMN
924D- A0 02           LDY #2             point at line #
924F- B1 9B           LDA (TPTR),Y
9251- 85 1D           STA LINNUM+1
9253- C8              INY
9254- B1 9B           LDA (TPTR),Y
9256- 85 1C           STA LINNUM
9258- 20 95 92        JSR PRINT.LINE.NUMBER
925B- A0 01           LDY #1             set up next pointer
925D- B1 9B           LDA (TPTR),Y
925F- F0 0B           BEQ .2             end of chain
9261- 48              PHA
9262- 88              DEY
9263- B1 9B           LDA (TPTR),Y
9265- 85 9B           STA TPTR
9267- 68              PLA
9268- 85 9C           STA TPTR+1
926A- D0 DE           BNE .1          ...always

926C- 60         .2   RTS
                 *-------------------------------
                 TAB.NEW.LINE
926D- 20 8E FD        JSR MON.CROUT
```

```
                TAB.NEXT.COLUMN
9270- A9 07      .1      LDA #7          first tab stop
9272- C5 24      .2      CMP MON.CH      cursor position
9274- B0 16              BCS .3          perform tab
9276- 69 06              ADC #6          next tab stop
9278- C9 21              CMP #33         end of line?
927A- 90 F6              BCC .2
927C- E6 16              INC COUNTER     count the screen line
927E- A5 16              LDA COUNTER
9280- 29 01              AND #1          look at odd-even bit
9282- F0 E9              BEQ TAB.NEW.LINE both scrn and printer
9284- A9 8D              LDA #CR
9286- 20 F0 FD           JSR MON.COUT1   <CR> to screen only
9289- 4C 70 92           JMP .1       ...always

928C- F0 06      .3      BEQ .4          already there
928E- E5 24              SBC MON.CH      calculate # of blanks
9290- AA                 TAX
9291- 20 4A F9           JSR MON.PRBL2
9294- 60         .4      RTS
                *--------------------------------
                PRINT.LINE.NUMBER
9295- A2 04              LDX #4          print 5 digits
9297- 86 1A              STX LZFLAG      turn on leading zero flag
9299- A9 30      .1      LDA #'0         digit=0
929B- 48         .2      PHA
929C- 38                 SEC
929D- A5 1C              LDA LINNUM
929F- FD D2 92           SBC PLNTBL,X
92A2- 48                 PHA
92A3- A5 1D              LDA LINNUM+1
92A5- FD D7 92           SBC PLNTBH,X
92A8- 90 0A              BCC .3          less than divisor
92AA- 85 1D              STA LINNUM+1
92AC- 68                 PLA
92AD- 85 1C              STA LINNUM
92AF- 68                 PLA
92B0- 69 00              ADC #0          increment digit
92B2- D0 E7              BNE .2       ...always

92B4- 68         .3      PLA
92B5- 68                 PLA
92B6- C9 30              CMP #'0
92B8- F0 0C              BEQ .5          zero, might be leading
92BA- 38                 SEC             turn off LZFLAG
92BB- 66 1A              ROR LZFLAG
92BD- 09 80      .4      ORA #$80
92BF- 20 ED FD           JSR MON.COUT
92C2- CA                 DEX
92C3- 10 D4              BPL .1
92C5- 60                 RTS
92C6- 24 1A      .5      BIT LZFLAG      leading zero flag
92C8- 30 F3              BMI .4          no
92CA- E0 00              CPX #0          if all zeroes, print one
92CC- F0 EF              BEQ .4
92CE- A9 20              LDA #'          blank
92D0- D0 EB              BNE .4       ...always

92D2- 01        PLNTBL   .DA #1
92D3- 0A                 .DA #10
92D4- 64                 .DA #100
92D5- E8                 .DA #1000
92D6- 10                 .DA #10000
92D7- 00        PLNTBH   .DA /1
92D8- 00                 .DA /10
92D9- 00                 .DA /100
92DA- 03                 .DA /1000
92DB- 27                 .DA /10000
                *--------------------------------
                CHECK.FOR.PAUSE
92DC- AD 00 C0           LDA KEYBOARD    keypress?
92DF- 10 11              BPL .2          no, go on
92E1- 8D 10 C0           STA STROBE
92E4- C9 8D              CMP #CR         RETURN?
92E6- F0 0A              BEQ .2          yes
92E8- AD 00 C0   .1      LDA KEYBOARD    no, wait for
92EB- 10 FB              BPL .1          another stroke
92ED- 8D 10 C0           STA STROBE
92F0- C9 8D              CMP #CR         return .EQ. if RETURN
92F2- 60         .2      RTS
```

Speaking of Slow Chips.......................Robert H. Bernard

William O'Ryan's article (AAL June 1984) about making the 65C02 work in II+s reminds me of some other slow chip problems I have had in the past with Apples.

Years ago, I had a problem with an SSM AIO card in an Apple that traced to a slow 74LS138 at position H2. The symptom was that every few hours the program would fly off into the weeds. I traced it to the device select for the slot, which caused the data on the bus to be late for ROM program fetches from the card. I was able to fix the problem in that case by swapping H2 with another '138 from a different (less critical) position.

Some time later I was able to fix a problem in another machine by swapping the ROM SELECT chip at position F12 (another 74LS138) with another '138. There are apparently many marginal timing situations in II+s, and they are not necessarily in the oldest ones.

All this slow circuit stuff has some interesting side effects. I personally had a number of conversations with SSM about this problem before I found the real cause, and all they could suggest was a capacitor on the clock line. Even after I found the problem, the SSM people I talked to seemed uninterested in the fix, perhaps because they couldn't apply it directly to their product.

The unfortunate end result was that a number of organizations that previously sold or recommended AIO cards stopped doing so. A domino effect was that our local retailer stopped pushing Anadex printers (which required the DTR signal, at that time only available on the AIO) rather than find another serial card to replace the AIO. I always wondered if the Anadex people noticed the effect on their sales....

Modify DOS 3.3 for Big BSAVEs..............Bob Sander-Cederlof

Jim Sather (author of "Understanding the Apple II" and designer of the QuikLoader card) called today, and one topic of discussion was DOS 3.3's limit of 32767 for the maximum size of a binary file. Jim has been blowing 27256 EPROMs, which are 32768 bytes long. To write a whole EPROMs worth of code on disk it takes two files, because the EPROM holds one more byte than the maximum size file.

The limit doesn't apply if you write the file with the .TF directive in the S-C Macro Assembler, but it is checked when you type in a BSAVE command. The "L" parameter must be less than 32768.

I remembered that somewhere very recently I had read of a quick patch to DOS to remove the restriction. Where? Hardcore Computing? Call APPLE? Washington Apple Pi?

The answer was "yes" to both Call APPLE and W.A.P., because Bruce Field's excellent Apple Doctor column is printed in both magazines. The July 1984 Call APPLE, on page 55, has the answer:

> "Sure, change memory location $A964 in DOS from $7F to $FF. From Applesoft this can be done with POKE 43364,255. This changes the range attribute table in DOS to allow binary files as large as 65535 bytes."

By the way, please do not try to BSAVE 65535 bytes on one file. You will not succeed, because doing so will involve saving bytes from the $C000-C0FF range. This is where all the I/O soft switches are, any you will drive your Apple and peripherals wild. And you will not be able to BLOAD it, because it will load on top of the DOS buffers. In general, do not BSAVE any area of RAM which includes $C000-C0FF. Do not BLOAD into the DOS buffers or DOS variables.

If you want to test Bruce's patch, make the patch and then BSAVE filename,A$800,L$8E00. This will save from $800 through $95FF.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).